プログラマーの本気が Excel を覚醒させる

超絶 Excel VBA　特別付録

4 時限でわかる VBA

1限目
基本文法

VBA の文法は非常に簡易です。初めてプログラミングに挑戦する人でも、比較的容易に習得が可能でしょう。ここでは VBA の文法を 10 分で確認できるよう最低限必要な項目をまとめてみました。

初めてプログラミングに挑戦するという方は、多少退屈でも実際に自分で構文を入力して動作を実際に確かめてみることをオススメします。有名なアインシュタインは「自分自身で経験する以上に良い方法はない」という名言を残していますが、プログラミングにおいてもこの名言が役立ちます。

文法の確認方法

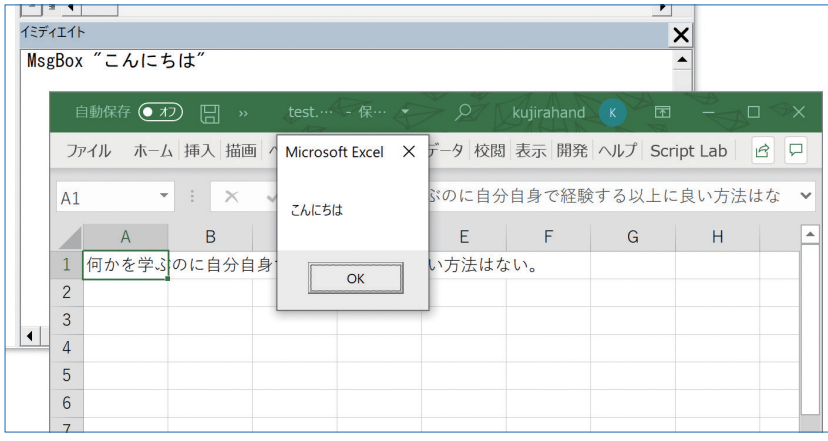
Excel で新規ブックを作成し、マクロ有効ブックの形式で「test.xlsm」などの適当な名前で保存しましょう。[Alt]+[F11] キーを押して VB エディターを表示しましょう。そして、プロジェクトエクスプローラーで Sheet1 をダブルクリックし、そこにサンプルプログラムを書いて実行しましょう。

また、簡単なプログラムであれば、メニューの [表示 > イミディエイト ウィンドウ] クリックしてイミディエイトウィンドウに直接プログラムを入力することでプログラムの結果を確認できます。

画面にメッセージを表示する

イミディエイトウィンドウに以下のように記述して実行していましょう。画面に「こんにちは」とメッセージボックスを表示できます。

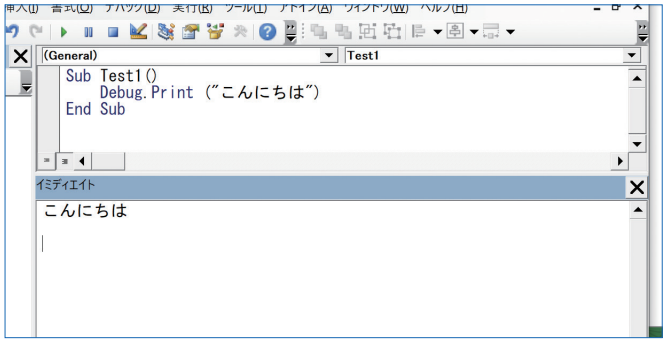
```
MsgBox "こんにちは"
```



▲ MsgBox を使ったところ

他には、Debug.Print を利用してイミディエイトウィンドウに値を出力できます。以下は、Sheet1 のコードウィンドウにプログラムを書いて実行してみてください。

```
Sub Test1()
    Debug.Print ("こんにちは")
End Sub
```



▲ Debug.Print を使ったところ

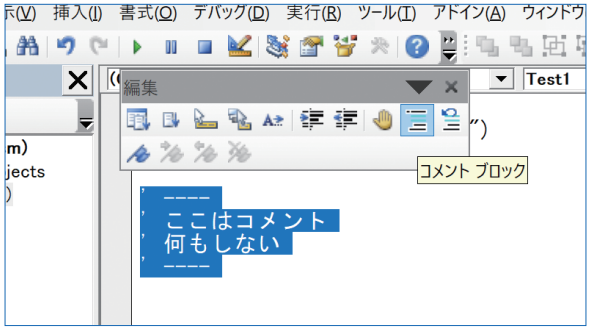
コメントについて

VBA では『'』(シングルクォート) から行末までがコメントです。コメントには何を書いても良くプログラムに影響を与えることはありません。プログラムの説明や注釈を書くのに利用します。

```
' ---- ---- ---- ---- ----
' ここはコメント
' 何もしない
```

```
' ---- ---- ---- ---- ----
```

なお、メニューの [表示 > ツールバー > 編集] で編集バーを表示すると、任意の選択範囲を一気にコメントに変換する「コメント」ボタンを表示できます。



▲ **一気に任意の範囲をコメントできる**

四則計算について

VBA ではいろいろな演算ができます。以下のような四則演算があります。

▼ **演算子の種類**

演算子	説明	利用例	例の結果
+	足し算	3 +	8
-	引き算	8 - 3	5
*	かけ算	2 * 3	6
/	割り算	9 / 4	2.25
^	累乗	2 ^ 3	8
Mod	剰余（割り算の余り）	9 Mod 4	1
&	文字列の足し算	100 & " 円 "	"100 円 "

イミディエイトウィンドウで、以下のように入力して四則計算を試してみると良いでしょう。なお、イミディエイトウィンドウでは、Debug.Print を "?" と簡略化して書くことができます。

```
Debug.Print 1 + 2 * 3 '← 式を入力
 7 ' ← 計算結果

? (1 + 2) * 3 '← 式を入力
 9 '← 計算結果
```

変数と代入

変数とは値をメモリに覚えておくための箱のようなものです。変数に値を代入するには、「変数名 = 値」と記述します。以下は、果物の値段を表す Apple や Banana という変数を利用して買い物の計算をするプログラムです。

```
Sub 買い物計算()
    Apple = 300
    Banana = 210
    Total = Apple * 3 + Banana * 5
    MsgBox "合計は" & Total & "円です"
    '結果→ 合計は1950円です
End Sub
```

初期状態では特に宣言することなく変数を利用できますが、変数を使うことを明示的に宣言できます。変数宣言は「Dim 変数名」あるいは「Dim 変数名 As 変数型」のように記述します。上記と同じプログラムを変数宣言してから使ってみます。

```
Sub 買い物計算_変数宣言()
    ' 変数を宣言する
    Dim Apple As Long
    Dim Banana as Long
    Dim Total as Long
    ' 変数に値を代入
    Apple = 300
    Banana = 210
    Total = Apple * 3 + Banana * 5
    MsgBox "合計は" & Total & "円です"
    '結果→ 合計は1950円です
End Sub
```

なお、VBA では変数名の書き間違いを防ぐために、変数宣言を強制できます。そのためにはプログラムの先頭に、『Option Explicit』と書きます。これがあると宣言なしで書いた変数をエラーにできます。つまり変数宣言を強制できます。ある程度規模のあるプログラムを作る時には『Option Explicit』を利用すると便利です。

また、変数名には英数字に加えて、漢字・ひらがな・カタカナなどのマルチバイト文字と『_』（アンダースコア）を利用できます。ただし、数字から始まる名前や VBA で使うキーワード、予約語（As や Boolean、Long、For など）は変数名に使えません。

データ型について

VBA で扱う値には、整数・実数・文字列などの型があります。以下のような型があります。

▼ VBA のデータ型

型名	型宣言	説明
ブール型	Boolean	True または False
整数	Integer	-32,768 ～ 32,767 の整数
長整数型	Long	-2,147,483,648 ～ 2,147,483,647 の整数
通貨型	Currency	-922,337,203,685,477.5808 ～ 922,337,203,685,477.5807 の固定小数点数
浮動小数点数型(単精度)	Single	実数を表現 (4 バイト)
浮動小数点数型(倍精度)	Double	実数を表現 (8 バイト)
日付型	Date	西暦 100 年 1 月 1 日から 9999 年 12 月 31 日の日付表現
文字列	String	" いろは " など文字列
オブジェクト型	Object	オブジェクト
バリアント型	Variant	あらゆるデータ型が代入できる

10 進数と 16 進数について

私たちの生活の中で使う数値は 10 進数が使われていますが、コンピューターに関連する計算では 16 進数が使えると便利です。10 進数では 9 の次が 10 ですが、16 進数では 9 の次が A です。そして、A,B,C,D,E,F となり、次が 10 となります。

16 進数を記述するには、&HFF のように『&H』から値を書きます。10 進数を 16 進数に変換するには『Hex(数値)』を使います。以下、イミディエイトウィンドウで 16 進数と 10 進数の変換を試すことができます。

```
? &H10
 16
? &HFF
 255
? Hex(16)
10
? Hex(255)
FF
```

配列変数について

1 つの変数の中に複数の変数を代入できるのが配列変数です。VBA で配列を変数には Dim を利用します。

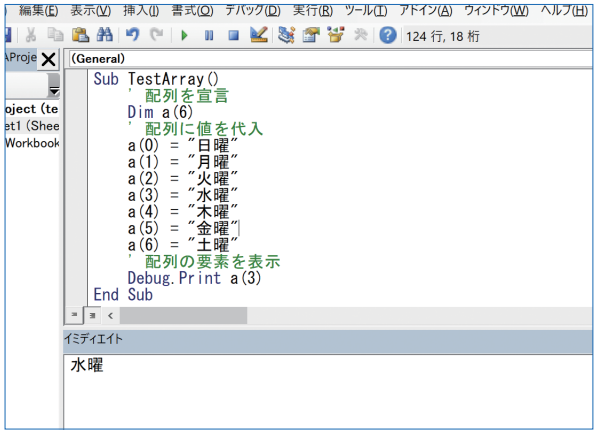
```
[書式]
' 配列変数の宣言
Dim 変数名(最大値)

' 配列変数に値を代入
変数名(要素番号) = 値
```

```
' 配列変数の値を参照
Debug.Print 変数名(要素番号)
```

簡単にプログラムを実行して動作を確かめてみましょう。以下のプログラムでは、配列変数 a を宣言し、配列 a の要素に日曜から土曜までの曜日を代入します。そうするなら、a(3) のように書くことで番号に応じた曜日を取得できます。

```
Sub TestArray()
    ' 配列を宣言
    Dim a(6)
    ' 配列に値を代入
    a(0) = "日曜"
    a(1) = "月曜"
    a(2) = "火曜"
    a(3) = "水曜"
    a(4) = "木曜"
    a(5) = "金曜"
    a(6) = "土曜"
    ' 配列の要素を表示
    Debug.Print a(3)
End Sub
```

プログラムを実行してみると、水曜が表示されます。



▲ **配列変数の利用例**

なお、配列変数を定義する際、明示的に要素番号の最低値と最大値を指定できます。もしも、a(5) のように要素番号の最小値を指定しなかった場合、最小値は 0 となります。つまり、a(5) と宣言した場合、a(0 To 5) と宣言したのと同じになります。

```
' 配列の宣言で0から5の配列変数の宣言
Dim N(5)
' 配列の宣言で1から8の配列変数を宣言
```

```
Dim M(1 to 8)
```

そして、VBA で宣言できるのは、上記のような要素が固定の配列だけではありません。ReDim を利用すると要素の最大値を変更できます。

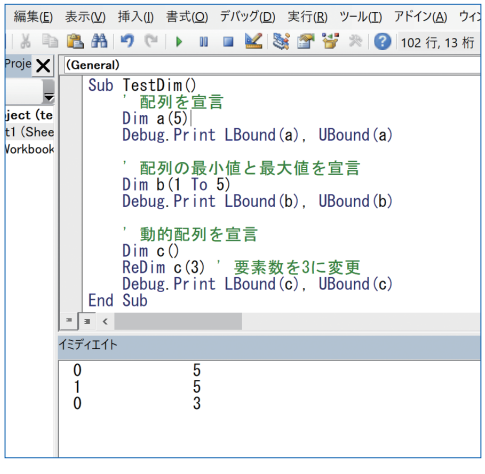
```
[書式]
' 動的配列を宣言
Dim L()
' 配列の要素数を最大8に変更する
Redim L(8)
```

配列の要素番号の最小値を調べるには LBound 関数、最大値を調べるのに UBound 関数が利用できます。

以下のプログラムは配列を宣言するプログラムです。

```
Sub TestDim()
    ' 配列を宣言
    Dim a(5)
    Debug.Print LBound(a), UBound(a)

    ' 配列の最小値と最大値を宣言
    Dim b(1 To 5)
    Debug.Print LBound(b), UBound(b)

    ' 動的配列を宣言
    Dim c()
    ReDim c(3) ' 要素数を3に変更
    Debug.Print LBound(c), UBound(c)
End Sub
```

上記のプログラムを実行すると、配列変数 a, b, c の要素番号の最小値と最大値を調べて表示します。

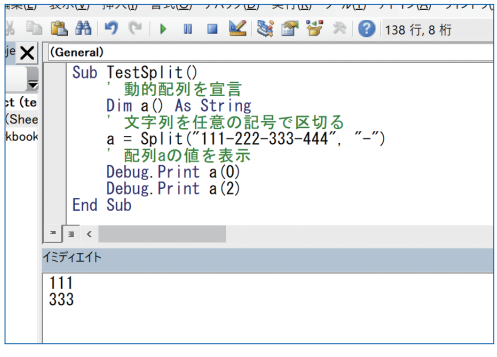


▲ 配列をいろいろ定義したところ

ここで動的配列を使う例を紹介します。文字列を任意の記号で区切る Split 関数を使うと任意の要素の配列を得ることができます。Split を使う前に配列の宣言をしますが、宣言の時点では、いくつ配列が必要になるのか分かりません。このような場合に動的配列を宣言します。

```
Sub TestSplit()
    ' 動的配列を宣言
    Dim a() As String
    ' 文字列を任意の記号で区切る
    a = Split("111-222-333-444", "-")
    ' 配列aの値を表示
    Debug.Print a(0)
    Debug.Print a(2)
End Sub
```

プログラムを実行すると、Split を利用して文字列「111-222-333-444」を「-」で分割します。すると分割された各要素が配列 a に代入されるので、要素番号を指定して要素を取り出して表示します。



▲ Split で文字列を分割する例

2限目
VBAの制御構文

プログラムを分岐させたり、繰り返したりする制御構文を学びましょう。

If 文 - 条件分岐

条件に応じてプログラムの流れを変える If 文があります。条件が偽の時の処理 (Else 以下) は省略できます。

```
[書式]
If 条件式 Then
    ' 条件が真の時の処理
Else
    ' 条件が偽の時の処理
End If
```

変数 n の値が偶数か奇数かを判定するプログラムは以下のように記述します。

```
Sub 偶数奇数判定()
    n = 11
    If n Mod 2 = 0 Then
        MsgBox n & "は偶数"
    Else
        MsgBox n & "は奇数"
    End If
End Sub
```

プログラムを [F5] キーで実行すると「11 は奇数」とメッセージボックスに表示されます。
また、複雑な条件を連続で判定していきたい場合には、ElseIf を利用して 1 つずつ条件を判定します。

```
[書式]
' 連続で条件判定する場合
If (条件式1) Then
    ' 条件式1に合致したときの処理
ElseIf (条件式2) Then
    ' 条件式2に合致したときの処理
ElseIf (条件式3) Then
    ' 条件式3に合致したときの処理
```

```
Else
    ' 上記のいずれにも合致しなかったときの処理
End If
```

比較式と論理演算子

If 文の条件式には以下の比較演算子を利用できます。

▼ **if 文で使える比較式の利用例**

比較式	意味
a = b	a が b と等しい
a <> b	a が b と異なる
a > b	a が b より大きい
a < b	a が b より小さい
a >= b	a が b 以上
a <= b	a が b 以下

また、論理演算子も利用できます。

▼ **if 文で使える論理演算子の利用例**

論理演算子	意味
a And b	論理積 (AND) a かつ b
a Or b	論理和 (OR) a または b
Not a	論理否定 (NOT) a ではない
Xor	排他的論理和 (XOR)

While 文 - 条件指定の繰り返し

条件を指定して繰り返しを行うには、while 文を使います。以下のような書式で指定します。

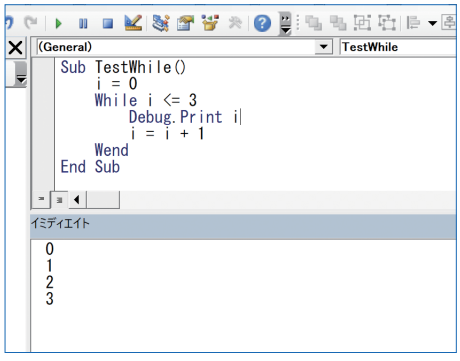
```
[書式]
While 条件式:
    ' 繰り返し行う処理
Wend
```

下記のプログラムは、0 から 3 まで繰り返し画面に表示するプログラムです。

```
Sub TestWhile()
    i = 0
    While i <= 3
        Debug.Print i
        i = i + 1
```

```
    Wend
End Sub
```

このプログラムを [F5] キーで実行すると以下のようにイミディエイトウィンドウに値が出力されます。



▲ While の利用例

Do While/Until ... Loop 文 - 条件指定の繰り返し

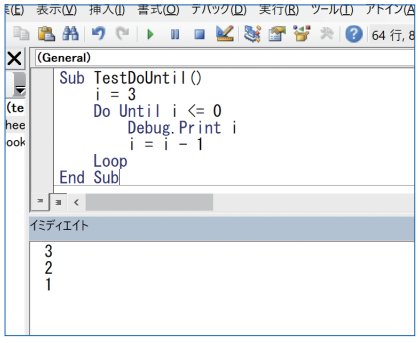
上記の While ... Wend 文は Do While ... Loop と同じ意味になります。まったく同じで 1，2，3 と表示します。

```
Sub TestDoWhile()
    i = 0
    Do While i <= 3
        Debug.Print i
        i = i + 1
    Loop
End Sub
```

これに対して、Do Until ... Loop は条件が真になるまでの間（つまり条件が偽の間）繰り返し処理を実行します。

```
Sub TestDoUntil()
    i = 3
    Do Until i <= 0
        Debug.Print i
        i = i - 1
    Loop
End Sub
```

上記プログラムを実行すると、i が 0 以下になるまで、条件が偽の間繰り返し処理を実行します。



▲ 条件が満たされるまで繰り返す Do Until 構文

この二つの構文を比較しながら書式を確認してみましょう。

```
[書式]
Do While (条件)
    // 条件が真の間
Loop

Do Until (条件)
    // 条件が偽の間
Loop
```

それぞれ、条件の真偽に応じて繰り返し処理を実行します。

For 文 - 繰り返し

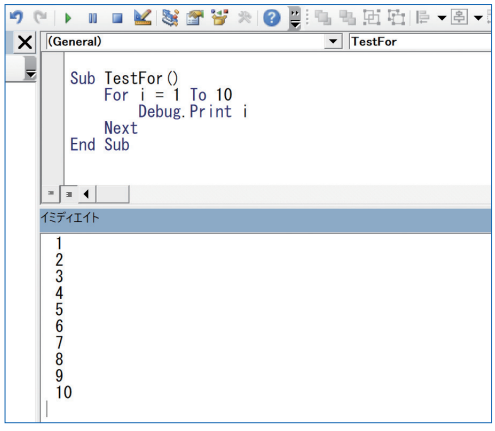
繰り返しを行うには、For 文を使います。以下のような書式で指定します。

```
[書式]
For 変数 = 開始値 To 終了値
    ' 繰り返し実行する処理
Next

For 変数 = 開始値 To 終了値 Step 増分値
    ' 繰り返し実行する処理
Next
```

以下は 1 から 10 までの数字をイミディエイトウィンドウに出力するプログラムです。

```
Sub TestFor()
    For i = 1 To 10
        Debug.Print i
    Next
End Sub
```

プログラムを [F5] キーで実行すると、イミディエイトウィンドウに数値が繰り返し出力されます。

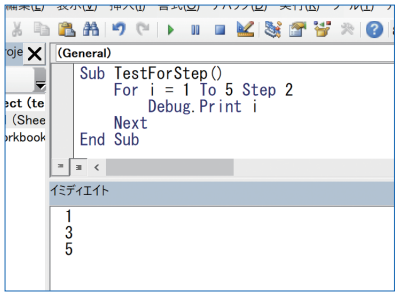


▲ 1 から 10 まで出力したところ

次に、For 文で Step を使った繰り返しを見てみましょう。For 文では開始値から終了値の間までを 1 ずつ増やしつつ繰り返すのですが、Step を指定すると指定した値分ずつ加算して繰り返します。

```
Sub TestForStep()
    For i = 1 To 5 Step 2
        Debug.Print i
    Next
End Sub
```

実行してみましょう。1 から 5 まで値を 2 ずつ増やして繰り返します。



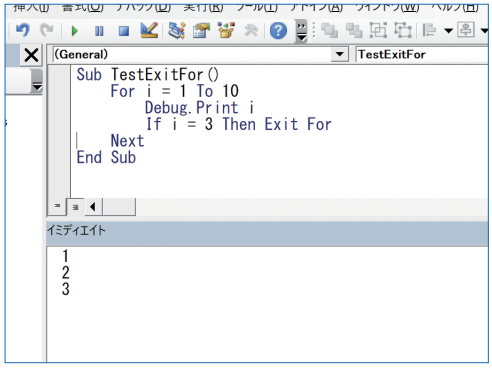
▲ For 文で Step を使ってみたところ

Exit For - 繰り返しからの脱出

繰り返しの途中で繰り返しを脱出したい場合があります。そのときは、Exit For を使います。以下のプログラムは、本来 1 から 10 まで繰り返すものですが、3 になったとき For 文から脱出します。

```
Sub TestExitFor()
    For i = 1 To 10
        Debug.Print i
        If i = 3 Then Exit For
    Next
End Sub
```

1 から 10 まで実行する For 文ですが、If 文と Exit For を記述することにより 3 回目で繰り返しを脱出します。上記のプログラムを実行すると以下のように表示されます。



▲ Exit For を使ったところ

Select Case 文 - 多分岐

If 文は真と偽でプログラムを挙動を変えますが、Select Case 文ではテストする値に応じて複数の分岐の中から合致する値を実行します。以下が Select Case 文の書式です。

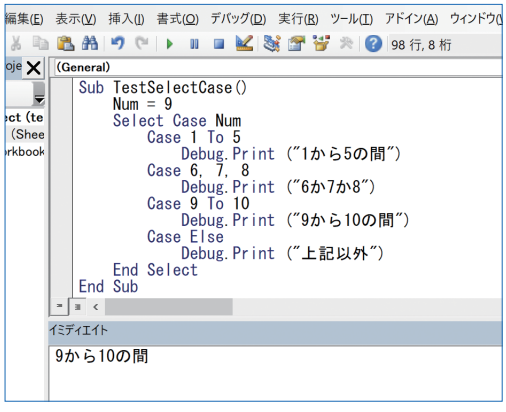
```
[書式]
Select Case テスト値
    Case 値1
        ' テスト値が値1だった時の処理
    Case 値2
        ' テスト値が値2だった時の処理
    Case 値3
        ' テスト値が値3だった時の処理
    Case Else
        ' テスト値が上記いずれにも該当しなかった時の処理
End Select
```

なお『Case』に続く値ですが『1 To 3』のような範囲を指定したり『1, 2, 3』のように複数の値を指定したりできます。実際に Select Case を使うプログラムを確認してみましょう。

```
Sub TestSelectCase()
    Num = 9
```

```
    Select Case Num
        Case 1 To 5
            Debug.Print ("1から5の間")
        Case 6, 7, 8
            Debug.Print ("6, 7, 8")
        Case 9 To 10
            Debug.Print ("9から10の間")
        Case Else
            Debug.Print ("上記以外")
    End Select
End Sub
```

プログラムを実行すると変数 Num の値である 9 に合致する選択肢を選んでプログラムを実行します。



▲ Select Case 文の利用例

3限目
プロシージャの定義

『プロシージャ』(procedure) とは、「手続き」という意味の英単語です。これは、一連の意味のある複数の処理に名前を付けておいて、別のプログラムから呼び出して使えるようにする仕組みです。

モダンなプログラミング言語で言うと、関数 (英語 :function) とかメソッド (英語 :method) と言い換えることができます。しかし、Excel にはシートのセルに書き込む関数もあります。そこでセルの数式で使う関数と区別する意味もあり、VBA 内の手続きは『プロシージャ』と呼ばれ戻りを持たないプロシージャを「Sub プロシージャ」、戻り値を持つプロシージャを「Function プロシージャ」と呼びます。

プロシージャについて

それでは VBA のプロシージャの書式を見てみましょう。戻り値があるかないかでプロシージャの定義方法が異なります。

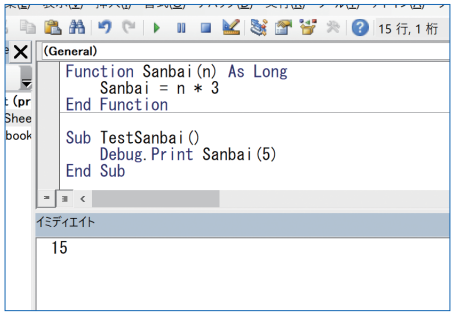
```
[書式]
' 戻り値を持たないプロシージャ
Sub プロシージャ名 ( 引数1, 引数2, 引数3, ... )
    ' プロシージャの定義
End Sub

' 戻り値を持つプロシージャ
Function プロシージャ名 ( 引数1, 引数2, 引数3, ... ) As 戻り値の型
    ' プロシージャの定義
   プロシージャ名 = 戻り値
End Sub
```

簡単なプロシージャの定義

簡単な例として引数に与えた値を三倍にする Function プロシージャの Sanbai を定義してみましょう。

```
' Sanbaiというプロシージャを定義
Function Sanbai(n) As Long
    Sanbai = n * 3
End Function

Sub TestSanbai()
    ' Sanbaiというプロシージャを呼び出す
    Debug.Print Sanbai(5)
End Sub
```

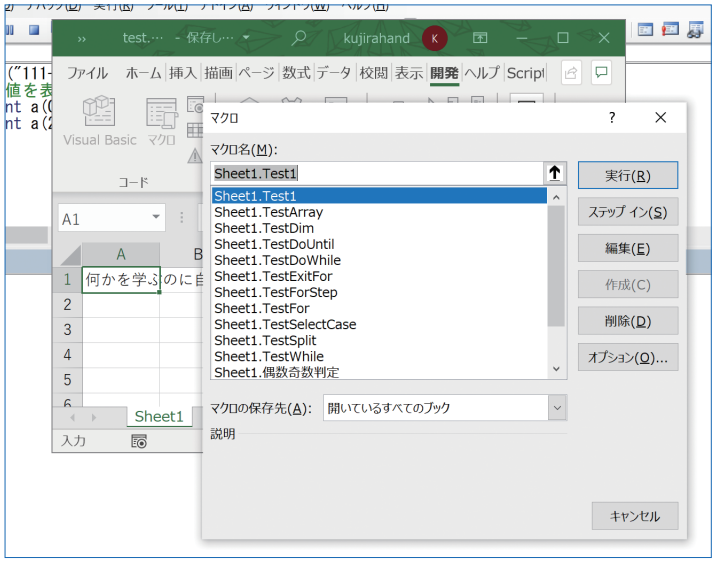
上記のプロシージャ TestSanbai を実行してみましょう。すると、値 5 を三倍にして 15 をイミディエイトダイアログに表示します。



▲ 値を 3 倍にするプロシージャを定義したところ

「マクロ」ダイアログに表示されるプロシージャ

なお、Sub プロシージャを定義すると VBA のマクロとして「マクロ」ダイアログから呼び出して実行させることが可能です。

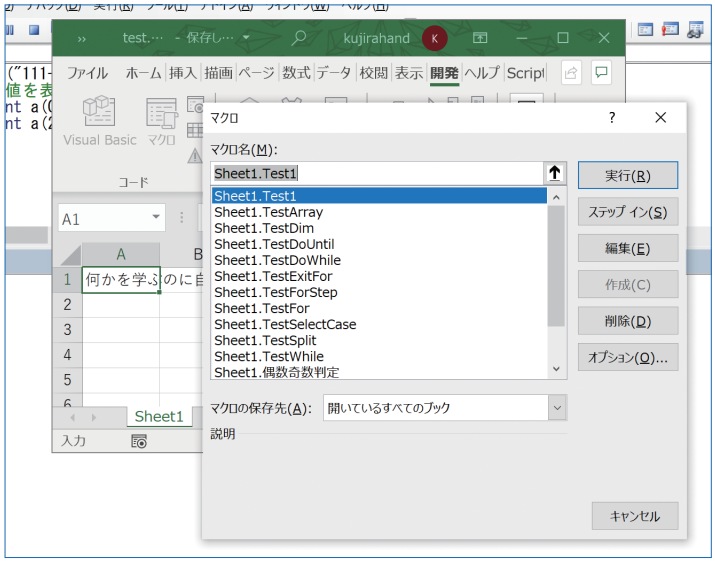


▲ Sub プロシージャはマクロとして呼び出せる

もしも「マクロ」ダイアログにプロシージャを表示したくない場合には、プロシージャの定義の定義を「Private Sub 名前」のように書きます。例えば、以下のように AAA と BBB というプロシージャを定義した場合、Private をつけた AAA はマクロダイアログに表示されません。

```
Private Sub AAA()
    MsgBox "AAA"
End Sub

Sub BBB()
    MsgBox "BBB"
End Sub
```

開発タブから「マクロ」をクリックするとマクロダイアログが表示されます。Private を付けたプロシージャは表示されません。



▲ Private を付けるとマクロダイアログに表示されなくなる

4限目

Excelワークシート・セル読み書きまとめ

VBA の文法の次に、Excel のワークシートの基本的な操作方法を紹介します。

ワークシートを選ぶ

最初に Excel のワークシートを選択する方法を確認してみましょう。ここでワークブック内にある Sheet1 を選択する方法を確認します。以下のようにいくつかの方法でシートを選択できます。

```
[書式]
' ワークブックのシート名からSheet1を選択 --- (※1)
Set sheet = ThisWorkbook.Sheets("Sheet1")

' ワークブックの先頭にあるシートを選択 --- (※2)
Set sheet = ThisWorkbook.Sheets(1)

' 直接 Sheet1を指定して選択 --- (※3)
Set sheet = Sheet1
```

(※ 1) はシート名から取得する方法、(※ 2) はシートの番号から取得する方法、(※ 3) は直接シート名を指定する方法となっています。なお、オブジェクトを変数に代入する場合、Set を利用して代入する必要があります。

セルの読み書き

次にセルの値を読み書きする方法を確認しましょう。セルを指定するには、セル名を指定する Range を使う方法、行番号・列番号を指定する Cell を使う方法と大きく言って 2 系統があります。

```
' 行番号と列番号から値を読む --- (※1)
Debug.Print sheet.Cells(1, 1)

' セル名から値を読む --- (※2)
```

```
Debug.Print sheet.Range("A1").Value

' 行番号と列番号から値を設定 --- (※3)
sheet.Cells(2, 1) = "猫に小判"

' セル名で値を設定 --- (※4)
sheet.Range("A2").Value = "猫に小判"
```

(※1)と(※3)が行番号と列番号からセルを指定する方法、(※2)と(※4)がセル名から値を指定する方法です。For 文などで連続で値を設定したい場合には前者の Cell を使う方法、特定のセルに値を設定するなら後者の Range を使う方法が便利です。

セルに計算式を指定する

そして、セルの値として「=」から始まる計算式や関数を指定すれば、計算を行うことができます。

```
Sub SetFormula()
    Dim sheet As Worksheet
    Set sheet = ThisWorkbook.Sheets(1)
    ' 数値をD列に設定 --- (※1)
    sheet.Range("D1").Value = 30
    sheet.Range("D2").Value = 50
    sheet.Range("D3").Value = 8
    ' 計算式を指定した --- (※2)
    sheet.Range("D4").Value = "=SUM(D1:D3)"
    ' 計算結果の88をイミディエイトウィンドウに出力 --- (※3)
    Debug.Print sheet.Range("D4").Value
    ' 計算式をイミディエイトウィンドウに出力 --- (※4)
    Debug.Print sheet.Range("D4").Formula
End Sub
```

プログラムを確認してみましょう。(※1)ではシートの D 列に 3 つ数値を設定します。そして(※2)でセルの値に「=」から始まる計算式を指定します。計算式ではセルを合計する SUM 関数を指定しています。(※3)ではセルの Value プロパティをイミディエイトウィンドウに計算済みの結果を表示します。これは計算済みの結果を出力します。これに対して(※4)では、Formula プロパティを参照しています。これは計算式そのものを得ます。

セルのプロパティやメソッドについて

なお、セルを操作するのに便利なプロパティやメソッドがあるので確認してみましょう。上記のように、Range で操作する範囲を指定した上で以下のプロパティやメソッドを利用します。『メソッド』とは実行することで何かしらの処理を行うものであり、『プロパティ』とは値の取得と設定ができるものです。以下はすべてではなく、よく使うものを抜粋したものです。

▼ よく使われるプロパティやメソッドの例

名前	種別	説明
Clear	メソッド	セルをクリア（数式・書式など全てクリア）
Value	プロパティ	セルの値（計算済みの値）
Formula	プロパティ	セルの値（計算式）
Interior .Color	プロパティ	RGB を使ってセル背景の塗り潰し色 (※ 1)
Font.Color	プロパティ	文字の色 (※ 1)
Font.Size	プロパティ	文字のサイズ
Copy	メソッド	セルをクリップボードにコピー
PasteSpecial（貼付タイプ）	メソッド	クリップボードを貼付け (※ 2)
Row	プロパティ	行番号
Column	プロパティ	列番号

(※ 1) 色の値を指定するには RGB（赤 , 緑 , 青）の値を指定します。

(※ 2) 貼付タイプに指定する値ですが、xlPasteValues で値だけ、xlPasteAll で書式も含めた全てを指定します。

Cells と Range の変換について

まず、A1 や D8 のようなセル名を行番号と列番号に変換してみましょう。そのためには、Range の Row と Column プロパティを利用します。

```
Sub RangeからCellsへの変換
    ' Rangeの行番号(8)を出力
    Debug.Print Sheet1.Range("D8").Row
    ' Rangeの列番号(4)を出力
    Debug.Print Sheet1.Range("D8").Column
End Sub
```

次に、行番号・列番号を指定してセル名を取得する方法を確認しましょう。方法は、Cells の Address メソッドを利用します。

```
Sub CellsからRangeへの変換()
    ' 行番号5, 列番号5のセル名 $E$5 を表示
    Debug.Print Sheet1.Cells(5, 5).Address
    ' 行番号5, 列番号5のセル名 E5 を表示
    Debug.Print Sheet1.Cells(5, 5).Address(False, Flase)
End Sub
```